

APPENDIX

```

#
#
# Value encodings:
5 # '' - String (and asciiz is not appended)
# "" - BMP string of ASCII charactes
# [] - Hex octet string
# <> - IP
# {} - Object ID
10 # Other - Integer

1 system = 0
2 allocations = 0
15 3 vtPoolSize = 100000
3 vtNodeCount = 30000
3 channels = 800
3 chanDescs = 5
3 messages = 400
20 3 nameChans = 10
3 tpktChans = 25
3 udpChans = 5
3 protocols = 800
3 maxProcs = 800
25 1 RAS = 0
2 responseTimeout = 20
2 allowCallsWhenNonReg = 0
2 manualRegistration = 0
30 2 manualDiscovery = 0
3 defaultGatekeeper = 0
4 ipAddress = 0
5 ip = [c8c8c8c8]
5 port = 1719
35 2 registrationInfo = 0
3 terminalType = 0
4 vendor = 0
5 vendor = 0
40 6 t35CountryCode = 11
6 t35Extension = 11
6 manufacturerCode = 11
5 productId = 'Test application'
5 versionId = 'RADVision'
45 4 terminal = 0
4 mc = 0
4 undefinedNode = 0
3 terminalAlias = 0
4 * = 0
50 5 e164 = '12345'

2 rasMulticastAddress = 0
3 ipAddress = 0
4 ip = <224.0.1.41>
55 4 port = 1718
2 rasPort = 0

1 Q931 = 0
2 manualAccept=0
60 2 responseTimeout = 150
2 connectTimeout = 500
2 callSignalingPort = 1720
2 maxCalls = 10

65 1 h245 = 0
2 masterSlave = 0

```

```

3   terminalType = 50
3   timeout = 100
2   capabilities = 0
3   timeout = 100
5   3   terminalCapabilitySet = 0
4       sequenceNumber = 0
4       protocolIdentifier = [00]
4       multiplexCapability = 0
5       h2250Capability = 0
10  6       maximumAudioDelayJitter = 60
6       receiveMultipointCapability = 0
7       multicastCapability = 0
7       multiUniCastConference = 0
7       mediaDistributionCapability = 0
15  8       * = 0
9       centralizedControl = 0
9       distributedControl = 0
9       centralizedAudio = 0
9       distributedAudio = 0
20  9       centralizedVideo = 0
9       distributedVideo = 0

6       transmitMultipointCapability = 0
7       multicastCapability = 0
25  7       multiUniCastConference = 0
7       mediaDistributionCapability = 0
8       * = 0
9       centralizedControl = 0
9       distributedControl = 0
30  9       centralizedAudio = 0
9       distributedAudio = 0
9       centralizedVideo = 0
9       distributedVideo = 0

35  6       receiveAndTransmitMultipointCapability = 0
7       multicastCapability = 0
7       multiUniCastConference = 0
7       mediaDistributionCapability = 0
8       * = 0
40  9       centralizedControl = 0
9       distributedControl = 0
9       centralizedAudio = 0
9       distributedAudio = 0
9       centralizedVideo = 0
45  9       distributedVideo = 0

6       mcCapability = 0
7       centralizedConferenceMC = 0
7       decentralizedConferenceMC = 0
50  6       rtcpVideoControlCapability = 0
6       mediaPacketizationCapability = 0
7       h261aVideoPacketization = 0

4       capabilityTable = 0
5       * = 0
55  6       capabilityTableEntryNumber = 7111
6       capability = 0
7       receiveAudioCapability = 0
8       g711Ul原因64k = 60
60  5       * = 0
6       capabilityTableEntryNumber = 7110
6       capability = 0
7       receiveAudioCapability = 0
65  8       g711Alaw64k = 60

5       * = 0

```

18

```

6      capabilityTableEntryNumber = 728
6      capability = 0
7      receiveAudioCapability = 0
8      g728 = 60
5
5      * = 0
6      capabilityTableEntryNumber = 261
6      capability = 0
7      receiveVideoCapability = 0
10     8      h261VideoCapability = 0
9      qcifMPI = 1
9      cifMPI = 1
9      temporalSpatialTradeOffCapability = 0
9      maxBitRate = 600
15     9      stillImageTransmission = 0
5
5      * = 0
6      capabilityTableEntryNumber = 263
6      capability = 0
20     7      receiveVideoCapability = 0
8      h263VideoCapability = 0
9      sqcifMPI = 1
9      qcifMPI = 1
9      cifMPI = 1
25     9      maxBitRate = 1000
9      unrestrictedVector = 0
9      arithmeticCoding = 0
9      advancedPrediction = 0
9      pbFrames = 0
30     9      temporalSpatialTradeOffCapability = 0
9      errorCompensation = 0
5
5      * = 0
35     6      capabilityTableEntryNumber = 7231
6      capability = 0
7      receiveAudioCapability = 0
8      g7231 = 0
9      maxAl-sduAudioFrames = 8
40     9      silenceSuppression = 0
5
5      * = 0 # Sequence
45     6      capabilityTableEntryNumber = 120 # INTEGER [1..65535]
6      capability = 0
7      receiveAndTransmitDataApplicationCapability = 0 # Sequence
8      application = 0
9      t120 = 0
50     10     separateLANStack = 0 # NULL [0..0]
8      maxBitRate = 1000 # INTEGER [0..4294967295]
5
5      capabilityDescriptors = 0
55     5      * = 0
6      capabilityDescriptorNumber = 0
6      simultaneousCapabilities = 0
7      * = 0
8      * = 7111
60     8      * = 7110
8      * = 7231
8      * = 728
7      * = 0
8      * = 261
65     8      * = 263
7      * = 0
8      * = 120

```

```

2  channels = 0
3  * = 0
5  4  name = 'g711Alaw64k'
4  4  dataType = 0
5  5  audioData = 0
6  6  g711Alaw64k = 60

10 3  * = 0
4  4  name = 'g711Ulaw64k'
4  4  dataType = 0
5  5  audioData = 0
6  6  g711Ulaw64k = 60

15 3  * = 0
4  4  name = 'g728'
4  4  dataType = 0
5  5  audioData = 0
20 6  g728 = 60

3  * = 0
4  4  name = 'g7231'
4  4  dataType = 0
25 5  audioData = 0
6  6  g7231 = 0
7  7  maxAl-sduAudioFrames = 8
7  7  silenceSuppression = 0

30 3  * = 0
4  4  name = 'h261VideoCapability'
4  4  dataType = 0
5  5  videoData = 0
6  6  h261VideoCapability = 0
35 7  qcifMPI = 1
7  7  cifMPI = 1
7  7  temporalSpatialTradeOffCapability = 0
7  7  maxBitRate = 600
7  7  stillImageTransmission = 0

40 3  * = 0
4  4  name = 'h263VideoCapability'
4  4  dataType = 0
5  5  videoData = 0
45 6  h263VideoCapability = 0
7  7  sqcifMPI = 1
7  7  qcifMPI = 1
7  7  cifMPI = 1
7  7  maxBitRate = 1000
50 7  unrestrictedVector = 0
7  7  arithmeticCoding = 0
7  7  advancedPrediction = 0
7  7  pbFrames = 0
7  7  temporalSpatialTradeOffCapability = 0
55 7  errorCompensation = 0

3  * = 0
4  4  name = 't120'
4  4  dataType = 0
60 5  data = 0
6  6  application = 0
7  7  t120 = 0
8  8  separateLANStack = 0
6  6  maxBitRate = 1000

65 [MWIACTIVATE_SR]

```

```

servedUserNr=serveduser@example.com
basicService=0815
msgCentreId=4711
nbOfMessages=1
5  originatingNr=originating@example.com
   timestamp=19970621194530
   priority=7

10  [MWIDEACTIVATE_SR]
   servedUserNr=serveduser@example.com
   basicService=0815
   msgCentreId=4711
   callbackReq=1

15  [MWIINTERROGATE_SR]
   servedUserNr=serveduser@example.com
   basicService=0815
   msgCentreId=4711
   callbackReq=1

20  [MWIACK_SR]
   basicService=0815
   msgCentreId=4711
   nbOfMessages=1
25  originatingNr=originating@example.com
   timestamp=19970621194530
   priority=7

30  [MWIREJ_SR]
   Error=8

35  #include <rvcommon.h>
   #include <msg.h>
   #include <ms.h>
   #include <pdlproc.h>
   #include <stkutils.h>
   #include <rtree.h>
   #include <pdlraw.h>
   #include <cmintr.h>
40  #include "h450api.h"
   #include "h450struct.h"

45  static LOCAL_STRUCT_PTR control;
   static cmElem* app;
   static char string[2048];      //string, which carries the H.450
   message
   static unsigned int n;          // counter to step through
   the string

50

RVAPI void CALLCONV initAPI(MYHANDLE *ahandle)
{
55     control = (LOCAL_STRUCT_PTR)calloc(1, sizeof(LOCAL_STRUCT));
    *ahandle = (MYHANDLE *)control;
}

60  RVAPI void CALLCONV endAPI()
    {
        if (control)
        {
            free(control);
        }
65  }

```



```

        // create new protocol: runs the create block of the specified
machine
    pdlCreateProtocol((HPdlSProtocol)app, (HPdlAProtocol)ahandle,
        (HPdlSProtocol*)h450shandle, machinename,
5    (newProtocolEH)newH450Child, (newMessageEH)newH450Message, FALSE);
    }

RVAPI void CALLCONV closeH450protocol(CALLBACK_T mglinkershandle)
10 {
    // close the protocol for the messagelinker
    pdlCloseProtocol((HPdlSProtocol)mglinkershandle);
}

15 int newH450Message(HPdlAProtocol haProtocol, HPdlSProtocol
hsProtocol, int message)
{
    HPST synH;
    HPVT valH;
20     int stNodeId;           // the syntaxtree nodeId
    int fieldId;             // the fieldId of the node
    int vtNodeId;            // the valuetree nodeId
    char rootname[256]; // the rootname of the syntaxtree root node
25     LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR)haProtocol;
    memset(string, 0, sizeof(string));

    valH = pdlGetValTree((HPdlSProtocol)hsProtocol);

    vtNodeId=pvtGetNodeIdByPath(valH, message, "");
30     synH=pvtGetSynTree(valH, vtNodeId);

    pvtGet(valH, vtNodeId, NULL, &stNodeId, NULL, NULL);
35     pstGetRootName(synH, sizeof(rootname), rootname);

    fieldId=pstGetFieldId(synH, rootname);
40     strcpy(string, "##### H450-Message
#####");
    local->callback(string);
    strcpy(string, "");
45     recursion(haProtocol, valH, synH, stNodeId, vtNodeId, fieldId,
0);

    // enable to send message in on esting: local-
55     >callback(string);

    return 0;

60 }

char recursion(HPdlAProtocol haProtocol, HPVT valH, HPST synH, int
stNode, int vtNode, INT32 fieldId, int tab)
65 {
    LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR)haProtocol;

```

```

    pstChild child;
    char fieldName[256];
    char fieldString[256];
    int stNodeIdChild;      // the syntaxtree nodeId of the Child
5    int vtNodeIdChild;     // the valuetree nodeId of the Child
    int value;              // the value of the number stored in
the node or the leghth of the string stored in the node
    BOOL isString;         // TRUE if a string is stored in the
10 node                    // counter

    vtNodeIdChild=vtNode;
    stNodeIdChild=stNode;

15    // get the value stored in the node
    prvGet (valH, vtNodeIdChild, NULL, NULL, &value, &isString);

    // insert the tabs
20    for (i=0 ; i<tab; i++)
    {
        strcat(string, "    ");
    }

25    // get the fieldname of the syntaxtree node
    pstGetFieldName(synH, fieldId, sizeof(fieldName), fieldName);

    strcat(string, "<");
    strcat(string, fieldName);
30    strcat(string, ">");

    if (value<=0)
    {
        // Begin: remove to send message in one string
35        local->callback(string);
        strcpy(string, "");
        // End: remove to send message in one string
    }

40    // reads out the value, if a value is stored in the valuetree
node
    if (value>0)
    {
45        // read out the string, if a string is stored
        in the node
        if (isString == TRUE)
        {
50            prvGetString(valH, vtNodeIdChild,
                value, fieldString);
            strcat(string, fieldString);
        }

        // read out the number, if a number is stored
55        in the node
        else
        {
            // convert number to string
            _itoa(value, fieldString, 10);
60            strcat(string, fieldString);
        }
        local->callback(string);
        strcpy(string, "");
65    }

    // if there are any children of the current node , the function

```



```

// calls itself with the stNode, vtNode and the fieldId of each
child, and with
// tab increased by one
5      if (pstGetNumberOfChildren(synH, stNodeIdChild)>0)
        {
            for (i=1; i<pstGetNumberOfChildren(synH,
10      stNodeIdChild)+1; i++)
                {
                    pstGetChild(synH, stNodeIdChild, i, &child );
// function only calls itself, when the child
exists in the valuetree
15      if (pvtGetChild(valH, vtNode, child.fieldId,
&vtNodeIdChild)>0)
                {
                    recursion(haProtocol, valH, synH,
child.nodeId, vtNodeIdChild, child.fieldId, tab+1);
20      }
                }
        }

// if NodeType is Sequence of or Set of, the function also
25      searches for
// children of the Sequence of XXX / Set of XXX node.
if (pstGetNodeType(synH, stNode)==19 || pstGetNodeType(synH,
stNode)==20)
30      {
        stNodeIdChild=pstGetNodeOfId(synH, stNode);
        vtNode=pvtChild(valH, vtNode);

        for (i=1; i<pstGetNumberOfChildren(synH,
35      stNodeIdChild)+1; i++)
            {
                pstGetChild(synH, stNodeIdChild, i, &child );
// function only calls itself, when the child
exists in the valuetree
40      if (pvtGetChild(valH, vtNode, child.fieldId,
&vtNodeIdChild)>0)
                    {
                        recursion(haProtocol, valH, synH,
45      child.nodeId, vtNodeIdChild, child.fieldId, tab+1);
                    }
            }
        }

// insert the tabs
50      for (i=0 ; i<tab; i++)
        {
            strcat(string, "    ");
        }

55      strcat(string, "<");
        strcat(string, fieldName);
        strcat(string, ">");

// Begin: remove to send message in one string
60      local->callback(string);
        strcpy(string, "");
// End: remove to send message in one string

return 0;
65  }

```

```

int newH450RootChild(
    IN      HPdlAProtocol      haProtocolParent,
    IN      HPdlSProtocol      hsProtocol,
    OUT     HPdlAProtocol*     lphaProtocol,
5    IN      char*             protocolName,
    OUT     newProtocolEH*     newSessionP,
    OUT     newMessageEH*     newCallMsgP,
    OUT     BOOL*              selfDestructing)
{
10    *newSessionP=(newProtocolEH)newH450RootChild;
    *newCallMsgP=newH450Message;
    *lphaProtocol=haProtocolParent;
    *selfDestructing=TRUE;
15    return 0;
}

int newH450Child(
20    IN      HPdlAProtocol      haProtocolParent,
    IN      HPdlSProtocol      hsProtocol,
    OUT     HPdlAProtocol*     lphaProtocol,
    IN      char*             protocolName,
    OUT     newProtocolEH*     newSessionP,
25    OUT     newMessageEH*     newCallMsgP,
    OUT     BOOL*              selfDestructing)
{
    *newSessionP=(newProtocolEH)newH450Child;
    *newCallMsgP=newH450Message;
30    *lphaProtocol=haProtocolParent;
    *selfDestructing=TRUE;
    return 0;
}

35
RVAPI void CALLCONV sendmessageH450(char * message, CALLBACK_T
h450shandle)
40 {
    int nodeId;
    cmElem*
app=(cmElem*)pdlGetAHandle(pdlRoot((HPdlSProtocol)h450shandle));

45    // get the handle of the valuetree
    app->hVal = pdlGetValTree(app->hsProtocol);

    // create a new value tree based on "h4507app2pdl"
    nodeId=pvtAddRoot(app->hVal,pdlGetSynTreeByRootName(app-
50 >hsProtocol,(char*)"h4507app2pdl"),0,NULL);
    n=0; /* set counter to 0*/

    // call function to fill in the fields of the valuetree sent in
    message
55    fillin(message, "", app->hVal, nodeId);
    pdlSendMessage((HPdlSProtocol)h450shandle, nodeId);
}

60 char fillin (char * message, char * rootpath, HPVT hVal, int nodeId)
{
    int length;
    int nodeIdTemp;
    char path[2048];
65    char fieldvalue[256];
    memset(path,0,sizeof(path));
    memset(fieldvalue,0,sizeof(fieldvalue));

```

```

strcpy(path, rootpath);
length = strlen(message);

5  while (message[n] == '<' && message[n+1] != '/')
    {
        n++;
        fillin(message, path, hVal, nodeId);
        if (message[n] == 0 || (message[n] == '<' && message[n+1] ==
10  '/')) return 0;
    }

    if (message[n] != '/')
    {
15        n--;
        if (n!=0) strcat(path, ".");
        n++;
        while (message[n] != '>')
        {
20            strncat(path, message+n, 1);
            n++;
        }
        n++;

25        if (message[n] != '<')
        {
            strcpy(fieldvalue, "");
            while (message[n] != '<')
            {
30                strncat(fieldvalue, message+n, 1);
                n++;
            }
            nodeIdTemp =
pvtBuildByPath(hVal, nodeId, path, sizeof(fieldvalue), fieldvalue);
35            n++;
        }

        else
        {
40            nodeIdTemp =
pvtBuildByPath(hVal, nodeId, path, 0, NULL);
            if (message[n] == '<' && message[n+1] != '/')
            {
                fillin(message, path, hVal, nodeId);
45                n++;
            }
            else n++;
        }
    }

50    if (message[n] == '/')
    {
        n--;
        while (message[n] != '>')
55        {
            n++;
        }
        n++;
        return 0;
60    }
}

#ifdef H450API_H
#define H450API_H

65 #include <cm.h>

```



```

    UINT32          key; /*protocol parameter is stored as tree
*/
    newProtocolEH    newProtocol; /* called when new child protocol
is created*/
5    newMessageEH    newMessage; /* protocol receives messages from
machine via this callback */
    BOOL            passive; /* set if tried to close protocol
and not succeded because of childs*/
    BOOL            selfDestructing; /* can parent close child
10    protocol without closing this child from
                                application*/
    } pdlElem; /*protocol structure */

/* Function prototypes */
15
int newH450RootChild(
    IN      HPdlAProtocol      haProtocolParent,
    IN      HPdlSProtocol      hsProtocol,
20    OUT     HPdlAProtocol*    lphaProtocol,
    IN      char*              protocolName,
    OUT     newProtocolEH*      newSessionP,
    OUT     newMessageEH*      newCallMsgP,
    OUT     BOOL*              selfDestructing

25    );

int newH450Child(
    IN      HPdlAProtocol      haProtocolParent,
    IN      HPdlSProtocol      hsProtocol,
30    OUT     HPdlAProtocol*    lphaProtocol,
    IN      char*              protocolName,
    OUT     newProtocolEH*      newSessionP,
    OUT     newMessageEH*      newCallMsgP,
35    OUT     BOOL*              selfDestructing

    );

40    int newH450Message(
        IN      HPdlAProtocol      haProtocol,
        IN      HPdlSProtocol      hsProtocol,
        IN      int                message
    );
45    char recursion(HPdlAProtocol haProtocol, HPVT valH, HPST synH, int
stNode, int vtNode, INT32 fieldId, int tab);
    char fillin (char * dummy, char * dummypath, HPVT hVal, int nodeId);

50    #endif /* H450_H */

#include <rvcommon.h>
#include <msg.h>
#include <ms.h>
55    #include <pdlproc.h>
#include <stkutils.h>
#include <rtree.h>
#include <pdlraw.h>
#include <cmintr.h>
60    #include <emanag.h>
#include "h450api.h"
#include "h450struct.h"

#include <ssintr.h>
65

static LOCAL_STRUCT_PTR control;

```

```

static ssElem* app;
static char string[2048];
static char buffer[2048];
static unsigned int n;

5
RVAPI void CALLCONV tpInitAPI(MYHANDLE *ahandle)
{
    control = (LOCAL_STRUCT_PTR) calloc(1, sizeof(LOCAL_STRUCT));
10    *ahandle = (MYHANDLE *) control;
}

RVAPI void CALLCONV tpEndAPI()
15 {
    if (control)
    {
        free(control);
20    }
}

RVAPI void CALLCONV tpSetCallback(MYHANDLE ahandle, CALLBACK_T
25 callback)
{
    LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR) ahandle;

    local->callback = callback;
30 }

RVAPI int tpNewH450Message(HPdlAProtocol haProtocol, HPdlSProtocol
hsProtocol, int message)
{
    HPST synH;
    HPVt valH;
    int stNodeId;
    int fieldId;
    int vtNodeId;
    char rootname[256];
    LOCAL_STRUCT_PTR local = (LOCAL_STRUCT_PTR) haProtocol;
40    memset(string, 0, sizeof(string));
    memset(buffer, 0, sizeof(buffer));

    valH = pdlGetValTree((HPdlSProtocol) hsProtocol);
45

    //vtNodeId=pvtGetNodeIdByPath(valH,
    message, "h4501SupplementaryService"); /* insert for h.450.1 */
    vtNodeId=pvtGetNodeIdByPath(valH, message, ""); /* remove for
    h.450.1 */
50

    synH=pvtGetSynTree(valH, vtNodeId);

    pvtGet(valH, vtNodeId, NULL, &stNodeId, NULL, NULL);

55    pstGetRootName(synH, sizeof(rootname), rootname);

    fieldId=pstGetFieldId(synH, rootname);

60    strcpy(string, "##### received H450-Message
#####"); /* remove for h.450.1 */
    local->callback(string); /* remove for h.450.1 */
    strcpy(string, "");

65    //emEncode((HVALT) valH, vtNodeId, (BYTE*) buffer, sizeof(buffer), &e
ncoded); /* insert for h.450.1 */
    //local->callback(buffer); /* insert for h.450.1 */

```



```

        {
            pvtGetString(valH, vtNodeIdChild, value,
fieldString);
            strncat(string, fieldString, value);
5            strcpy(fieldString, "");
        }

        else
10        {
            _itoa(value, fieldString, 10);
            strcat(string, fieldString);
            strcpy(fieldString, "");
15        }
        local->callback(string);
        strcpy(string, "");
    }
    else
20    {
        local->callback(string);
        strcpy(string, "");
    }
25    }

    if (pstGetNumberOfChildren(synH, stNodeIdChild)>0)
30    {
        for (i=1; i<pstGetNumberOfChildren(synH,
stNodeIdChild)+1; i++)
        {
35            pstGetChild(synH, stNodeIdChild, i, &child );
            if (pvtGetChild(valH,
vtNode, child.fieldId, &vtNodeIdChild)>0)
            {
40                tpReadOutVT(haProtocol, valH, synH,
child.nodeId, vtNodeIdChild, child.fieldId, tab+1);
            }
        }
45    }

    if (pstGetNodeType(synH, stNode)==19 || pstGetNodeType(synH,
stNode)==20)
50    {
        stNodeIdChild=pstGetNodeOfId(synH, stNode);
        vtNode=pvtChild(valH, vtNode);

        for (i=1; i<pstGetNumberOfChildren(synH,
55 stNodeIdChild)+1; i++)
        {
            pstGetChild(synH, stNodeIdChild, i, &child );
            if (pvtGetChild(valH,
60 vtNode, child.fieldId, &vtNodeIdChild)>0)
            {
                tpReadOutVT(haProtocol, valH, synH,
child.nodeId, vtNodeIdChild, child.fieldId, tab+1);
            }
65        }
    }

```



```

        for (i=0 ; i<tab; i++)
        {
            strcat(string, "    ");
        }
5      strcat(string, "</");
      strcat(string, fieldName);
      strcat(string, ">");
10     local->callback(string);
      strcpy(string, "");

      return 0;
15 }

RVAPI void CALLCONV tpSendMessageH4507(char * message, HSSSERVICE
h450shandle)
{
20     int nodeId;
    ssElem*
app=(ssElem*)pdlGetAHandle(pdlRoot((HPdlSProtocol)h450shandle));

    app->hVal = pdlGetValTree(app->hsProtocol);
25     nodeId=pvtAddRoot(app->hVal,pdlGetSynTreeByRootName(app-
>hsProtocol,(char*)"h4507app2pdl"),0,NULL); /* remove for h.450.1 */

    n=0; /* set counter to 0*/
30

    tpPopulateVT(message, "", app->hVal, nodeId);
    pdlSendMessage((HPdlSProtocol)h450shandle, nodeId);
}
35

RVAPI void CALLCONV tpSendMessageH4501(char * message, HSSSERVICE
h450shandle)
{
40     int nodeId;
    ssElem*
app=(ssElem*)pdlGetAHandle(pdlRoot((HPdlSProtocol)h450shandle));

    app->hVal = pdlGetValTree(app->hsProtocol);
45

    nodeId=pvtAddRoot(app->hVal,pdlGetSynTreeByRootName(app-
>hsProtocol,(char*)"h4501AppPrimitive"),0,NULL); /* insert for
h.450.1 */
50

    n=0; /* set counter to 0*/

    tpPopulateVT(message, "", app->hVal, nodeId);
55     pdlSendMessage((HPdlSProtocol)h450shandle, nodeId);
}

RVAPI void CALLCONV tpSendMessageH450x(char * message, HSSSERVICE
h450shandle)
60
{
    int nodeId;
    ssElem*
65     app=(ssElem*)pdlGetAHandle(pdlRoot((HPdlSProtocol)h450shandle));

    app->hVal = pdlGetValTree(app->hsProtocol);

```

```

5      n=0; /* set counter to 0*/

      tpPopulateVT(message, "", app->hVal, nodeId);
      pdlSendMessage((HPdlSProtocol)h450shandle, nodeId);
10  }

int chrn2bmp1(char *str, int maxStrLen, BYTE* bmpStr)
{
15     int i, i2;

    for (i = 0, i2 = 0; i < maxStrLen; i++, i2 += 2)
    {
        bmpStr[i2 + 0] = 0;
        bmpStr[i2 + 1] = str[i];
20     }

    return i2;
}

25 int chr2bmp1(char *str, BYTE* bmpStr)
{
    return chrn2bmp1(str, strlen(str), bmpStr);
}

30 char tpPopulateVT (char * message, char * rootpath, HPVT hVal, int
nodeId)
{
    UINT32 fieldvalueOctet;
    int length;
    int fieldvaluesize;
    int fieldvalueInt;
    int fieldvaluelength;
    int bmpLength;
    int nodeIdTemp;
40     char bmpStr[512];
    char path[2048];
    char fieldvalue[256];
    char stringOrInt;
45     memset(path,0,sizeof(path));
    memset(fieldvalue,0,sizeof(fieldvalue));
    strcpy(path, rootpath);
    length = strlen(message);
50     while (message[n] == '<' && message[n+1] != '/')
    {
        n++;
        tpPopulateVT(message, path, hVal, nodeId);
        if (message[n] == 0 || (message[n] == '<' && message[n+1] ==
55     '/') return 0;
    }

    if (message[n] != '/')
60     {
        n--;
        if (n!=0) strcat(path, ".");
        n++;
        while (message[n] != '>')
65     {
            strncat(path, message+n, 1);
            n++;

```



```

        return 0;
    }
}

5  #ifndef H450API_H
    #define H450API_H

    #include <cm.h>
10  #include <pdlapi.h>
    #include <h450.h>

    /* Typedefs */

15  typedef void *MYHANDLE;

    typedef void (*CALLBACK_T)(char*);

    typedef struct
20  {
        CALLBACK_T callback;
    } LOCAL_STRUCT, *LOCAL_STRUCT_PTR;

25  RVAPI void CALLCONV tpInitAPI(MYHANDLE *ahandle);
    RVAPI void CALLCONV tpEndAPI();
    RVAPI void CALLCONV tpSetCallback(MYHANDLE ahandle, CALLBACK_T
30  callback);
    RVAPI void CALLCONV tpSendMessageH4501(char * string, HSSSERVICE
        h450shandle);
    RVAPI void CALLCONV tpSendMessageH450x(char * string, HSSSERVICE
        h450shandle);
    RVAPI void CALLCONV tpSendMessageH4507(char * string, HSSSERVICE
35  h450shandle);
    RVAPI int tpNewH450Message(HPdlAProtocol haProtocol, HPdlSProtocol
        hsProtocol, int message);
    RVAPI int CALLCONV tpCreateService(HSSAPP hSSApp, HSSSERVICE*
        hSSServ, HSSAPPSERVICE hSSaServ, char* serviceName);

40  #endif /* H450API_H */
    #include <windows.h>
    static HINSTANCE hInst;
    HINSTANCE getHInst(void)
45  {
        return hInst;
    }

    #ifdef WIN32

50  BOOL WINAPI DllMain(
        IN HINSTANCE hinstDLL, // handle to DLL module
        IN DWORD fdwReason, // reason for calling function
        IN LPVOID lpvReserved // reserved
55  )
    {
        switch(fdwReason)
        {
            case DLL_PROCESS_ATTACH:
                hInst=hinstDLL;
                break;
            case DLL_PROCESS_DETACH:
                break;
        }
65  return TRUE;
    }

```

```

#else
int FAR PASCAL LibMain(HANDLE hInstance, WORD wDataSegment,
                        WORD wHeapSize, LPSTR lpszCmdLine)
5  {
    if (hInst != NULL)
        return FALSE;

    hInst = hInstance;
10  return TRUE;
}

int FAR PASCAL __export WEP (int bSystemExit)
15 {
    hInst = NULL;
    return 0;
}

#endif
20 #ifndef H450_H
#define H450_H

#include <cm.h>

25 char tpReadOutVT(HPdlAProtocol haProtocol, HPVt valH, HPST synH, int
stNode, int vtNode, INT32 fieldId, int tab);
char tpPopulateVT (char * dummy, char * dummypath, HPVt hVal, int
30 nodeId);

#endif /* H450_H */

```